International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 4, Issue 2, Apr 2014, 213-222

TRANS STELLAR
• Journal Publications • Research Consultancy

ESPY & ALLAY OF SECURITY VULNERABILITIES IN WEB-APPLICATIONS BASED ON JAVA PLATFORM

MOHIL PANDEY[1], UMESH KULKARNI[2], SUMIT BHATTACHARJEE[3] & AMOL YADAV[4]

[1,3,4]Department of Computer, MGM College of Engineering and Technology, Navi Mumbai, Maharashtra, India

[2]Department of Computer, Vidyalankar Institute of Technology, Mumbai, Maharashtra, India

ABSTRACT

Several inbuilt security features are directly offered by the Java programming language along with various characteristics supplied by several securities relevant APIs and implementations. Hence, directly choosing Java will not ensure complete security against severe secrecy attacks, integrity or availability attacks. Numerous security issues needs to be focused and implemented during the entire software development process, entirely free from the selected programming language. The paper reviews discuss all the facets of "Java Security" and then consider the numerous "vulnerabilities" & "secure programming techniques" which should be acknowledged when Java programming comes in action. Numerous vulnerabilities classifications and the corresponding intrusions are covered along with some allay (mitigation) techniques. The paper discusses the input data causing attack and how to prevent it.

KEYWORDS: Java Security, Software Vulnerabilities, Source Code Analysis, Resource Injection, Path Manipulation Security

INTRODUCTION

The scenarios of security vulnerabilities are changing fiercely in the past few decades. Although string format violations and overflowing of buffer memory resulted in a huge section of all exploited susceptibilities during late s1980s, the entire scenario started too diverse in the new millennium's latter years. The applications based on Web were becoming flashier, familiar overflow of buffer have been reduced by the vulnerabilities of Web application like cross-site scripting and SQL injections intrusions. Such vulnerabilities are mainly responsible for a slew of attacks across big financial institutions, e-commerce sites, and other sites, causing millions of dollars in amends. [1]

BACKGROUND

In the last two decades, web applications have emerged from simple, fixed pages to mosaic, full-fledged active applications. Usually, such applications are crafted from heterogeneous technologies and include code which runs on the client (example is JavaScript) and the one running on the server (example is Java Servlet). Nowadays, the simple web applications may obtain and operate multiple distinct HTTP parameters to be able to serve users with affluent, collective services. Thereby resulting in active web applications including ample area for validation of numerous input vulnerabilities like SQL injection, cross site scripting etc. [2] Sadly, because of their high popularity and a user base that consists of millions of Internet users, web applications have become prime targets for attackers.

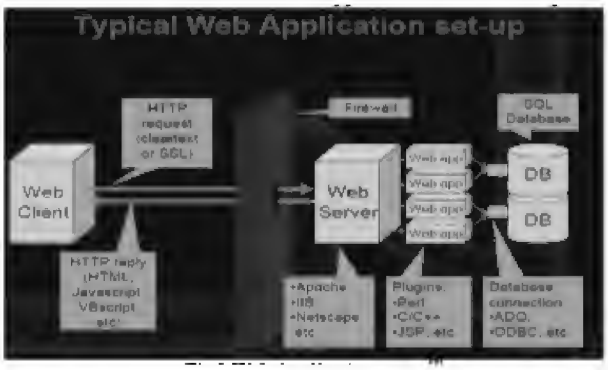

Figure 1: Web Application Set-Up

THE SOFTWARE DEVELOPMENT LIFE CYCLE 'S SECURITY

Since the developmental life cycle of software gets split in numerous manners, it commonly consists of the listed phases, which the developers can repeat, continuously: initialization and analysis, design and specification, implementation i.e. coding, testing of phases and code, execution, and decommissioning. Since the developers should consider security issues of code throughout the software development phases, they should specially keep tab of the key phases (three):

- **Implementation**

While coding, developers should adhere to best rules which neglects the most serious vulnerabilities in the definite application domain. An example rule consists of input and output validation, the detection of suspicious characters, and the application of parameterized instructions. Since these techniques are commonly efficient in neglecting utmost Web apply security vulnerabilities, the developers may not always employ them or it wrongly sometimes due to lack of knowledge related to security.

- **Testing**

Multiple methods are available for analyzing security vulnerabilities from testing, dynamic analysis, static analysis and runtime deviation detection. Issues with the developers majorly eye on testing functional essentials and apathy security facet. Moreover, current automated tools normally serve bad results such as either low susceptibility detection coverage or plenty of bogus positives.

- **Deployment**

During runtime multiple possibilities exists for inclusion of varying intrusion detection tools in the surroundings. These tools can function at distinct levels and practice several detection paths. Difficulties in usage may describe the accomplishments overhead and to the bogus positives which disturbs the normal working behavior of system. [4]

SECURE PROGRAMMING GUIDELINES

Programming securely is not feasible without following few general best programming rules. Therefore instructions are split into two segments. The first section covers regular rules which need to be pursued to write programs more securely, where as the latter segment focuses on specific topics of Java. The guidelines provided in the earlier section are somewhat common in nature and identical rules can be defined for other programming languages too. [5][6]

- **Guidelines for Security [5] [6]**
  - Keep it Simple
  - Only as Secure as the Weakest Link
  - Validate Input and Output.
  - Least Privilege
  - Defense in Depth
  - Fail Securely (Closed)
  - Compartmentalization (Separation of Privileges)
  - Use and Reuse Trusted Components
  - Security by Obscurity Won't Work
- **Java Specific Guidelines [5] [6]**
  - Garbage Collection
  - Exception Handling
  - Serialization and Deserialization
  - Java Native Interface (JNI)

VULNERABILITIES

An attacker, the "Threat" can exploit Vulnerability, which is a security bug in the software. Altogether it seems to be a Risk. [5] In the following we define and describe common categories of Web Vulnerabilities. [7]

Code Injection (COD)

Cookie Security (COO)

Cross Site Scripting (XSS)

Flow Injection (FLO)

Information Disclosure (INF)

Input Validation (INP)

Path Traversal (PAT)

Resource Injection (RES)

SQL Code Injection (SQL)

Unreleased Resources (UNR)

Logic Errors (LOG)

Table 1: Web Application Vulnerability Category [7]

Category Vulnerability	Descriptions	Related to (attacks)
Code Injection (COD)	• The injection of system and script commands into a web application or an application's server. • This kind of attack mostly applies to server side script languages like PHP or Perl.	INP PAT RES
Cookie Security (COO)	• This category includes several security vulnerabilities based on cookies, e.g., unfiltered cookie content, cookie poisoning, and flow injection via cookies. • In a broader sense, this section is related to session management.	INP
Cross Site Scripting (XSS)	• Here, the attacker inserts code into a URL or link. • The malicious URL must be executed by a web application's user to have an effect. • Misleading users to execute such URLs is supported by the URL itself which looks like a trustworthy URL to the application. • This only works when the application is vulnerable to XSS. • The result can be, e.g., the execution of malicious script (e.g., JavaScript) commands on the client side.	INP
Directory Browsing	Path Traversal	
Directory Traversal	Path Traversal	
Flow Injection (FLO)	• It is a special case of logic errors and is usually not detectable by security scanners. • This vulnerability is based on setting application states which depend on untrustworthy user data. • Thus, the control flow of an application's code could be influenced by an attacker.	LOG
Information Disclosure (INF)	• An information disclosure security flaw can be defined as the emission of data or information which is not intended to become available to the public. • This can be internal or private data. • There are several issues in this category which are not only programming errors, like the wrong or public storage of sensitive data.	Information Disclosure (INF)
Input Validation (INP)	• Usually any input/external data – not only from users – of an application has to be checked to see whether it conforms to intended formats or properties. • Such procedures usually also involve data filtering (sanitization) and adequate output encoding. • If input validation, filtering, and output encoding are missing or incomplete, this can enable a variety of attacks.	COD COO RES SQL XSS
Logic Errors (LOG)	• All programming errors, but also errors in system design or specification, which cannot be classified in another security category are called logic errors. • Thus, these errors are not typical programming errors. • Moreover, it is usually not possible to test for resulting security flaws.	FLO
Path Browsing	see Path Traversal	
Path Traversal (PAT)	• Can be generally defined as unintended access to application files or directories by injecting (sub) paths and filenames. • The injection, for instance, can take place into application URLs.	COD INP RES
Category Vulnerability	• Definition for the area of IT security	Related to (attacks)
Resource Injection (RES)	• Resource injection flaws can be defined as a category of security vulnerability related to unintentional access to system resources via the application layer, like in the case of path traversal.	COD INP PAT
SQL Code Injection (SQL)	• Results of successful attacks of this category are the execution of arbitrary SQL statements and commands on the application's database backend(s).	INP
Unreleased Resources (UNR)	• Some program resources, which are, e.g., variables and class instances (objects), have to be explicitly unloaded for freeing application memory. • If they are not released properly and not caught by the Java garbage collector, they might lead to increased memory consumption. • Thus, in a broader sense, unreleased resources can enable "Denial of Service" attacks and are a concern for an application's security.	Unreleased Resources (UNR)

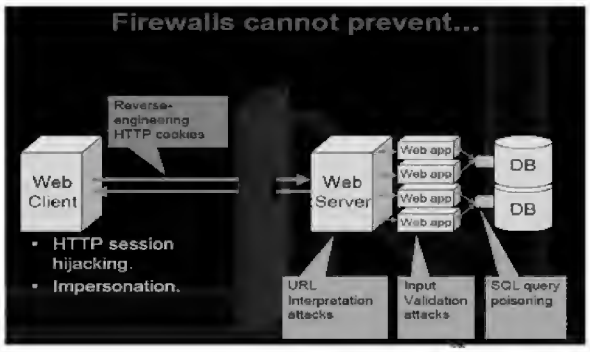

Figure 2: Web Application Set-up with Firewall

DETECTING SUSCEPTIBILITIES

Analyzing security concerns needs not only concentrating on verifying the application's functionalities along with detecting hazardous unseen flaws in the code which intruders can exploit. The two main ways for finding susceptibilities are white-box scrutiny and black-box testing. [4]

- **White-Box Testing**

White-box scrutiny includes examination of codes beyond executing it. Developers can proceed in one of two approaches: manually, while inspecting codes and reviews; or automatically, utilizing automated analysis tools. Code inspection involves a programmer's peers properly testing the delivered code, looking for programming errors. Security inspections are powerful approach for minimizing susceptibilities in the software. Nevertheless, such checkups are time consuming, expensive, and require thorough concepts of Web security. Alternative for this is inexpensive code review way, a simplified detection is useful for determining less severe code. Manually reviews can also be done, thereby excluding a formal detection meeting. Numerous experts do it individually along with moderator filters thereby clubbing the outcomes. Although an effective way, code review still seems to be quite expensive. Hence, for reducing the cost, developers sometimes depend on automated tools like static code analyzers. Such analyzers vet code, either in binary or source form, attempting to detect normal implementation-level faults. The main issue is that exhaustive scrutiny is complex and cannot detect many security errors due to lack of a dynamic (runtime) view and source code's complexity.

- **Black-Box Testing**

It refers to the scrutiny of program execution from an outsider's point of view. Shortly includes comparing software execution outcome with the expected outcome. The most powerful technique for software validation and verification is testing. Numerous manners exist for applying black-box testing, covering from unit to integration and then system testing. Such approach can be formal or less formal sometimes. The target of reliability testing, a special type of black-box testing that characterizes the system's behavior in the existence of erroneous input constraints. A special type of robustness testing named Penetration testing which scrutinizes execution of program in the presence of suspicious inputs, looking for possible vulnerabilities. Here, testers apply fuzzy techniques inclusive of submitting invalid or unexpected data, to a Web application and analyses its responses via HTTP requests. Numerous tests can reach multiples for each vulnerability type.

- **Limitations of Vulnerability Detection**

Static code analysis and Penetration testing can be automatic or manual. Because manual inspections or tests

require specifically resources of security making it time-consuming, automated tools for web application are choice of many developers. A vital fact while taking the restrictions of vulnerability detection tools for security testing is complex. Indeed, checking an application's security is a big question: although detecting few susceptibilities can be easy whereas guaranteeing that the application free of vulnerabilities is difficult. Both static code analysis and penetration testing tools have intrinsic constraints. The former relies on efficient code execution; however, practically, susceptibility identification only vets the Web application's result. On the other hand, the latter part can be complex. Code difficulty and the lack of a dynamic (runtime) scene may prevent detection of several security flaws. Although, penetration testing does not need access to the code whereas static code analysis does. Making use of wrong tools for detection may result in deployment of applications with unrevealed susceptibilities.

DETECTING ATTACKS

Intrusion detection includes identifying changes from learned behavior. Intrusion detection tools use methods based on either signatures or anomaly detection.

The needed fields for an efficient investigation are enlisted below

- URI requested
- HTTP Method
- Timestamp
- Full HTTP data sent
- Source IP

Intrusion Data could be in

- Cookie
- HTTP headers from client
- URI (uniform resource identifier)
- Basically anywhere

Techniques for Detection

Using static techniques

- Aim is forensics investigation
- Parse log files using standard tools/techniques
- Happens post-occurrence of event

Using dynamic techniques

- Trigger alarms when attack is happening
- Detect the attack as it happens

- Goal is to detect/prevent in real-time

Techniques for Static Detection

Sources of data to look at

- Operating system logs
- Web Application's custom audit trail
- Web Server Logs
- Application Server Logs

Missing terms:

- HTTP Headers only partially represented
- POST data (only GET data available)
- Cookie or Referrer data depends on web server

Static Detection Fails to Detect

Web application attacks in a POST form

- Cross-site scripting
- SQL injection

HTTP Header attacks can't be traced:

- Intrusions which makes overflow possible with HTTP header fields

HTTP header fields

- The Template of attack can't be detected

Forceful Browsing: user trial for accessing page excluding prior pages which would ensure correct authorization and authentication.

Static Detection Does Detect

- Authentication brute-forcing attacks
- Intrusions that check for server misconfigurations.
- HTML hidden field attacks (only if GET data –rare)
- Automated attacks using tools such as Nessus or Whisker or Nikto
- Order ID brute-forcing intrusions mainly if it is POST data, then order IDs cannot be seen

Techniques for Dynamic detection

Methods:

- Network-based IDS (possibly) adapted for applications
- In-line Application IDS
- Application Firewall

Advantages:

- POST request data URI request data
- Including HTTP headers
- Complete packet headers and payload available.

The intrusion detection space is fragmented into two possibilities:

- Anomaly-based
- Signature-based

Each has its own effectiveness and implementation issues.

Table 2: Anomaly Based Approach vs Signature Based

Anomaly-Based	Signature-Based
More complicated	Easier to implement
Mostly commercial solutions	Cheaper to modify, without expert help
False positives are fewer	False positives more
As well as false negatives	False negatives as well
Used for both web server, as well as web application attacks	Popular for detecting known web server attacks. Can be tweaked to do decent web application detection.

TYPE OF ATTACKS

- **Inject Suspicious Data into Java Applications**
  - Parameter manipulation: specially developed suspicious data values in HTML forms.
  - URL tempering: using specially designed parameters for submission as part of the URL on the Web application.
  - Hidden field tempering: fix unseen fields of HTML forms in Web pages to suspicious data values.
  - HTTP header manipulation: tamper bits of HTTP requests forwarded to the application.
  - Cookie poisoning [5]: place suspicious data values in cookies which are mini files forwarded to Web-based applications.
- **Manipulate Applications using Malicious Data.**
  - SQL injection: pass input data having SQL commands for execution to a database server.
  - Cross-site scripting: feat applications that output rampant input exactly to deceit the user into executing suspicious scripts.
  - HTTP response splitting: deed applications that output input rampant to achieve Web page vandalism or Web

cache poisoning intrusions.

- Path traversal: exploit unchecked user input to control which files are accessed on the server.
- Command injection: exploit user input to execute shell commands. [7]

CONCLUSIONS

JavaScript are being exploited to wreak havoc on the user's browser and operating system without even disturbing the policies of security of the browser. Normal code can be written which consumes memory or other resources of system and crashing the browser and even sometimes the operating system entirely. Further deceptive programming practices needs to get employed to trick or annoy the user in actions which is not intended by them. So the applications accepting user input data need to be aware for properly validating data's before accepting it, and to sanitize it finding embedding it into a Web page. The failure may result in cross-site scripting susceptibilities that are harmful like violations of the same origin policy would be. Responsibility lies with individual developers to write careful, clean code which would improve the user experience thereby causing them to lookout for suspicious users trying to let go their checks. Also the Software should be compounded with counter measures for above listed attacks. This can allow for expanded features. It can also reduce post attack coding efforts.

REFERENCES

1. Whitepaper Secure Programming in Java (c) 2005, EUROSEC GmbH Chiffriertechnik & Sicherheit.
2. S. Institute. Top Cyber Security Risks, September 2009. http://www.sans.org/ top-cyber-security-risks/summary.php
3. www.net-square.com,saumil@net-square.com, top ten web attacks.
4. Defending against Web Application Vulnerabilities, Nuno Antunes and Marco Vieira, University of Coimbra, Portugal, Published by the IEEE Computer Society, 0018-9162/12/$31.00 © 2012 IEEE,
5. www.cgisecurity.com.
6. Security Code Guidelines, Sun Microsystems, Inc. (http://java.sun.com/security/seccodeguide.html).
7. Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
8. Finding Security Vulnerabilities in Java Applications with Static Analysis, V. Benjamin Livshits and Monica0 S. Lam Computer Science Department Stanford University {livshits, lam}@cs.stanford.edu.
9. A Process for Performing Security Code Reviews, published by the IEEE computer society ■ 1540-7993/06/$20.00 © 2006 IEEE ■ IEEE security & privacy.